Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт
(государственный университет)»

Факультет управления и прикладной математики

Кафедра информатики

СРАВНЕНИЕ АЛГОРИТМОВ ОПТИМИЗАЦИИ «ЧЕРНОГО ЯЩИКА» И РЕАЛИЗАЦИЯ АЛГОРИТМА НА ОСНОВЕ ГАУССОВСКИХ ПРОЦЕССОВ ДЛЯ ОПТИМИЗАЦИИ МАШИННОГО ОБУЧЕНИЯ

Выпускная квалификационная работа
(дипломная работа)

Направление подготовки: 03.03.01 Прикладная математика и физика

Выполнил:
студент 176 группы _____ Шмелев Артем Олегович

Научный руководитель:
к.ф.-м.н., доцент _____ Устюжанин Андрей Евгеньевич

Москва 2015

Аннотация

Тема данной дипломной работы «Сравнение алгоритмов blackbox оптимизации и реализация алгоритма Gaussian Processes для оптимизации машинного обучения» в рамках которой были рассмотрены существующие алгоритмы blackbox оптимизации, выполнена реализация одного из алгоритмов Gaussian Processes. Используя этот алгоритм была сделана оптимизация алгоритма grid search машинного обучения.

Объем дипломной работы 36 страниц, она содержит 21 рисунок, 9 таблиц, 8 источников литературы.
Ключевые слова: blackbox-оптимизация, гауссовские процессы, grid search, машинное обучение.
Дипломная работа состоит из введения, 4 глав и заключения.

В первой главе (введение) обоснована актуальность темы, поставлена цель написания дипломного проекта, определены предмет исследования, задачи.

Во второй главе рассматриваются существующие алгоритмы blackbox оптимизации, описывается реализация алгоритма Gaussian Processes (COGP), которая была выполнена.

В третьей главе приводится сравнение работы алгоритмов.

В четвертой главе рассматривается алгоритм grid search, оптимизация этого алгоритма с использованием COGP. Приводится сравнение работы этих алгоритмов.

В пятой главе демонстрируется результат работы оптимизированного алгоритма на данных Хиггса.

Заключение содержит основные выводы по эффективности использования алгоритма COGP и версии grid search с оптимизацией.

Содержание

1 Введение **4**

2 Обзор алгоритмов blackbox оптимизации **6**

2.1 Обзор существующих алгоритмов 6

2.2 Алгоритм Gaussian Processes (COGP) 7

3 Сравнение работы алгоритмов **9**

3.1 Функция $Branin$ 10

3.2 Функция $xsinx$ 18

3.3 Функция $Goldstein - Price$ 23

3.4 Функция $Colville$ 28

3.5 Вывод 28

4 Использование COGP для оптимизации grid search **30**

4.1 Схема работы grid search 30

4.2 Оптимизация с использованием COGP 30

4.3 Сравнение работы с классическим grid search 31

4.4 Вывод 32

5 Результат работы оптимизации на данных Хиггса **33**

5.1 Higgs Boson Machine Learning Challenge 33

5.2 Оптимизация метрики AMS 33

6 Заключение **35**

Список литературы **36**

1 Введение

В последнее время все большую популярность приобретает машинное обучение. Человек стремится переложить выполнение многих действий на машину, автоматизировать свой труд. При работе с большим количеством данных машина действительно успешно обучается и способна показать прекрасный результат.
Термин «машинное обучение» появился еще в 50-х годах. Этот термин обозначает попытку научить компьютер решать задачи, которые легко даются человеку, но формализовать путь их решения сложно. В результате машинного обучения компьютер может демонстрировать поведение, которое в него не было явно заложено.
Общую постановку задачи обучения можно сформулировать следующим образом. Имеется множество *объектов* и множество возможных *ответов*. Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая *обучающей выборкой*. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определенным образом вводится функционал качества.
С бурным развитием компьютерных технологий резко возросло число задач, которые машинное обучение с успехом решает и где оно играет ключевую роль. Сюда можно отнести:

- распознавание речи, жестов, рукописного ввода;
- компьютерное зрение;
- медицинская диагностика;
- техническая диагностика;
- обнаружение мошенничества, спама;
- кредитный скоринг;
- рекомендательные системы;
- автоматический перевод текста;
- ранжирование поисковых запросов.

Машинное обучение — обширный подраздел искусственного интеллекта, математическая дисциплина, использующая разделы математической статистики, численных методов оптимизации, теории вероятностей, дискретного анализа, выделяющая знания из данных. В данной работе рассматривается связь машинного обучения и методов оптимизации. Предметом исследования работы являются алгоритм машинного обучения *grid search* («поиск на сетке») и его улучшение с помощью алгоритма из методов оптимизации.
Методы оптимизации — важный раздел математики, информатики и исследования операций. Задачи оптимизации широко встречаются в физике (динамика твердого тела), экономике, машинном обучении. Алгоритмы оптимизации — это средства, решающие эти задачи.

Стандартная математическая задача оптимизации формулируется таким образом. Среди элементов x, образующих множество X, найти такой элемент $\hat{x}$, который доставляет минимальное значение $f(\hat{x})$ заданной функции $f(x)$. На практике, как правило, стараются использовать эффективные алгоритмы оптимизации. Поэтому является важным время выполнения алгоритма и качество (точность) полученного результата.
В работе затрагиваются алгоритмы *blackbox* оптимизации. *Blackbox* оптимизация (оптимизация «черного ящика») — это поиск экстремума функции, которая не задана аналитически. Функция словно *черный ящик*: как она устроена, характер ее поведения — неизвестны, но, подав на вход *ящику* точку, тут же будет получено значение в этой точке.
Главными задачами данной работы являются:

- сделать краткий обзор существующих алгоритмов оптимизации «черного ящика»;
- собрать информацию и реализовать *blackbox* алгоритм на основе гауссовских процессов;
- сравнить реализованный алгоритм с другими;
- придумать способ оптимизировать алгоритм машинного обучения *grid search* при помощи реализованного алгоритма;
- реализовать оптимизацию, сделать сравнение оптимизированного варианта алгоритма и обычного.

Главная же цель — попытаться улучшить алгоритм *grid search*, который широко используется в машинном обучении. *Grid search* позволяет выбирать оптимальные параметры для тренировки обучаемого алгоритма. Как и любой другой алгоритм машинного обучения он работает на большом наборе данных, и, следовательно, его слабое место — это время выполнения. Поэтому, в первую очередь, улучшение алгоритма должно решать эту проблему.

2 Обзор алгоритмов blackbox оптимизации

2.1 Обзор существующих алгоритмов

В работе сделан краткий обзор существующих алгоритмов blackbox оптимизации. Рассматривались следующие алгоритмы: NelderMead, CMAES, HillClimber, GA, OriginalNES, ExactNES, FEM.

NelderMead

Метод Нелдера Мида, также известный как метод деформируемого многогранника. Суть метода заключается в последовательном перемещении и деформировании симплекса (n-мерное обобщение треугольника) вокруг точки экстремума.

GA

Генетический алгоритм. Метод имитирует естественную эволюцию. Воспроизводит популяцию искомых точек. Из этих точек «выживают» точки с наибольшим значением функции. Затем выжившие точки участвуют в воспроизведении потомства. Воспроизведение выполняется с помощью мутаций и скрещиваний. Их реализация зависит от конкретного варианта алгоритма.

HillClimber

На вход алгоритм принимает стартовую точку, которую он считает текущим решением. На каждом шаге алгоритм ищет точку, в которой значение функции больше. Точка ищется вблизи текущего решения, небольшим его изменением. Если точка улучшает значение функции, она принимается алгоритмом и становится текущим решением. Когда улучшающей точки не найдено, экстремум найден и алгоритм завершается.

CMAES

CMAES Covariance Matrix Adaptation Evolution Strategy («адаптация ковариационной матрицы»). Относится к классу эволюционных алгоритмов. В отличие от генетического алгоритма моделирует популяцию искомых точек по заданному распределению (многомерному нормальному) и не использует для этого прошлое поколение точек. На каждом шаге обновляется ковариационная матрица для более точного приближения к экстремуму.

OriginalNES

NES Natural Evolution Strategies. Так же, как и CMAES, моделирует точки по заданному распределению. На каждом шаге обновляет параметры распределения, сдвигая их вдоль естественного градиента (natural gradient), который вычисляется по значениям в точках, матрице Фишера и текущим параметрам распределения.

ExactNES

Более новая версия (2009) алгоритма OriginalNES. Использует точное выражение для вычисления информационной матрицы Фишера вместо приближения.

FEM

FEM — Fitness Expectation-Maximization. Алгоритм схож с NES и CMAES, но использует явный вид распределения, а не параметризованный. На каждом шаге распределение обновляется согласно принципу expectation-maximization, т.е. максимизируется матожидание значения функции $\mathbb{E}_{\mathbf{x}}[f(\mathbf{x})]$.

2.2 Алгоритм Gaussian Processes (COGP)

Для реализации и более детального рассмотрения был выбран алгоритм оптимизации «черного ящика» Gaussian Processes. Другое название — COGP (Curiosity-driven optimization with Gaussian processes). Алгоритм относится к классу Curiosity-driven optimization. Основной принцип алгоритмов этого класса — «принцип любознательности». Любознательность, тяга к неизведанному позволили человечеству исследовать окружающий мир, способствовали развитию представлений о мире и привели к тому пониманию Вселенной, которое сейчас есть. Этот общий прием можно применить и в гораздо более узкой задаче — поиск экстремума функции. На каждом шаге поиска максимума при выборе очередной точки алгоритм стремится выбрать наиболее информативную, наиболее интересную точку (дающую наибольший прирост информации).

Однако, полагаясь только на этот принцип и выбирая всегда самую информативную точку, алгоритм не гарантирует, что точки будут близки к максимуму или будут сходится к максимуму (не гарантирует максимального *fitness improvement*). Таким образом, в алгоритме должен учитываться и второй принцип: нужно стараться выбирать точку ближе к текущему максимуму. Получается, что алгоритм должен решать задачу двухцелевой оптимизации.

Чтобы использовать алгоритм на практике, нужно иметь явные выражения для функции прироста информации и функции fitness improvement. В работе [6] показано, что в качестве функции ожидаемого fitness improvement можно взять:

$$\Delta(x) = \sigma(Y|y_o)(s\Phi(s) + \phi(s)),$$

где

$$s = \frac{f_{max} - \mathbb{E}[Y|y_o]}{\sigma(Y|y_o)}, \tag{1}$$

Φ и ϕ — функции нормального распределения и плотности нормального распределения, Y — случайная величина, y_o — наше текущее знание, $\mathbb{E}$ — математическое ожидание, σ — среднеквадратическое отклонение, f_{max} — текущее максимальное значение.

В работе [1], используя гауссовские процессы, удается вывести простое выражение для ожидаемого прироста информации в точке x:

$$\Psi(x|y_o) = \frac{1}{2}\log\sigma^2(Y|y_o) - \frac{1}{2}\log\sigma_n^2. \tag{2}$$

(Полагается, что ковариационная функция гауссовского процесса имеет вид $k(x, x') + \sigma_n^2\delta(x, x')$, где $\delta(\cdot,\cdot)$ — символ Кронекера). Таким образом, максимизация ожидаемого информационного прироста — это просто максимизация условной дисперсии.

Algorithm 1 Псевдокод алгоритма COGP

Require: целевая функция f, начальные точки X

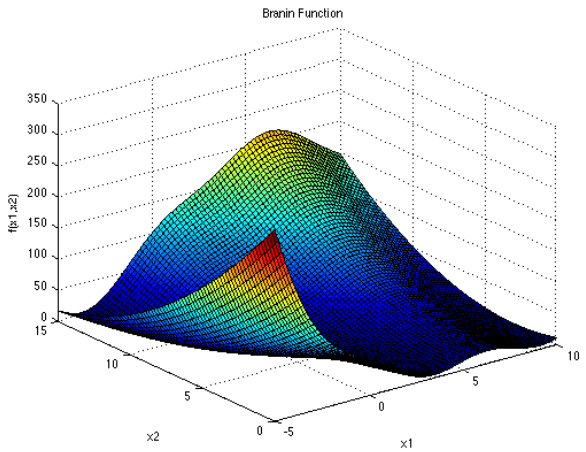
```
repeat
    Приближаем точки X гауссовским процессом G
    Находим множество C точек-кандидатов, максимизирующих Ψ_G(x) и Δ_G(x)
    Выбираем x* из C
    X ← X ∪ {(x*, f(x*))}
until критерий остановки выполнен (сделано нужное число шагов)
```

При реализации этого алгоритма была встречена трудность: согласно алгоритму на каждом шаге нам нужно, помимо приближения гауссовским процессом, решать задачу двухцелевой оптимизации — нужно максимизировать одновременно две функции. С одной стороны мы должны выбрать точку, в которой прирост информации будет наибольшим, с другой — мы должны брать точку, чтобы максимизировать fitness improvement, т. е. максимизировать матожидание значения функции. Максимизируя одну функцию, мы можем уменьшать другую. В работе [4] рассматривается данная проблема. Говорится, что при решении такой задачи оптимизации каждой функции можно приписать вес, однако, в зависимости от этих весов, вообще говоря, зависит и результат оптимизации. Таким образом, задав веса постоянными величинами, алгоритм подвергается тому, что в одних областях будет полностью доминировать одна функция, в других — другая функция. В данной работе веса функций при решении задачи двухцелевой оптимизации константно заданы. Для большинства функций, на которых производилось тестирование алгоритма, это не повлияло на правильность работы.

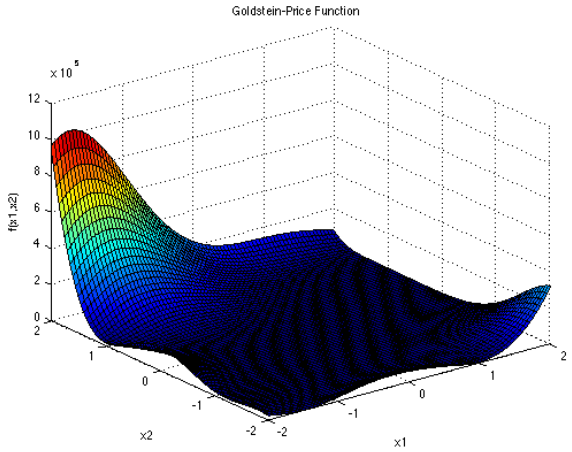
3 Сравнение работы алгоритмов

Для сравнения работы алгоритмов, они запускались, минимизируя 4 разные функции с разным числом шагов (числом выбираемых точек). Затем вычислялось отклонение вычисленного алгоритмом минимума от истинного минимума (ошибка). В качестве функций были выбраны:

- функция Branin: $f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)cos(x_1) + s$,
  где $a = 1, b = \frac{5}{4\pi^2}, c = \frac{5}{\pi}, r = 6, s = 10, t = \frac{1}{8\pi}$
- функция $f(\mathbf{x}) = xsinx$
- функция Goldstein-Price: $f(\mathbf{x}) = [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]\times$
  $[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
- функция Colville: $f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 +$
  $+10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$



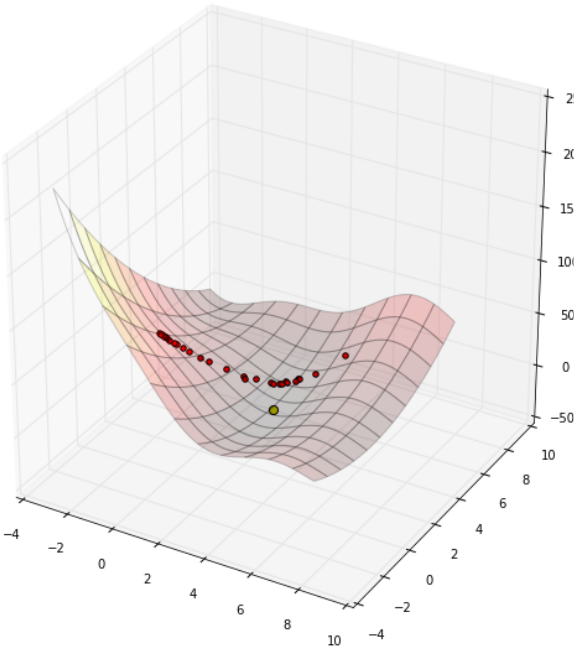
Branin



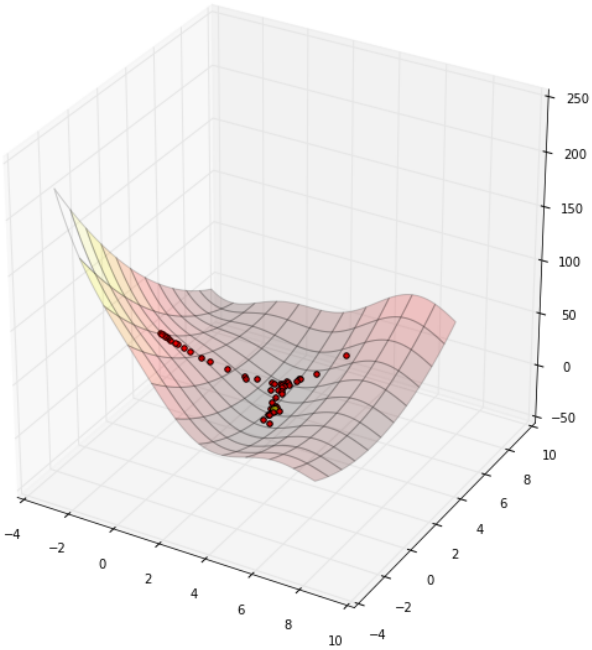
Goldstein-Price

Все функции, кроме $xsinx$ классические функции оптимизации. Они достаточно сложно устроены, чтобы экстремум нельзя было просто найти. На этих функциях часто проверяют работу различных оптимизационных алгоритмов.
Использовавшуюся реализацию алгоритма COGP, который был реализован в рамках данной работы, можно найти в репозитории *https://github.com/artshmelev/project-nir*. Реализация остальных алгоритмов была взята из библиотеки PyBrain.
Результаты были получены на компьютере Intel Core i5 2.67GHz 4Gb RAM.
Графики работы алгоритмов изображены на рис. 1-20. Желтый маркер обозначает положение истинного минимума, красные выбираемые алгоритмом точки.
Результаты сравнения оформлены в виде таблиц (табл. 1-4).

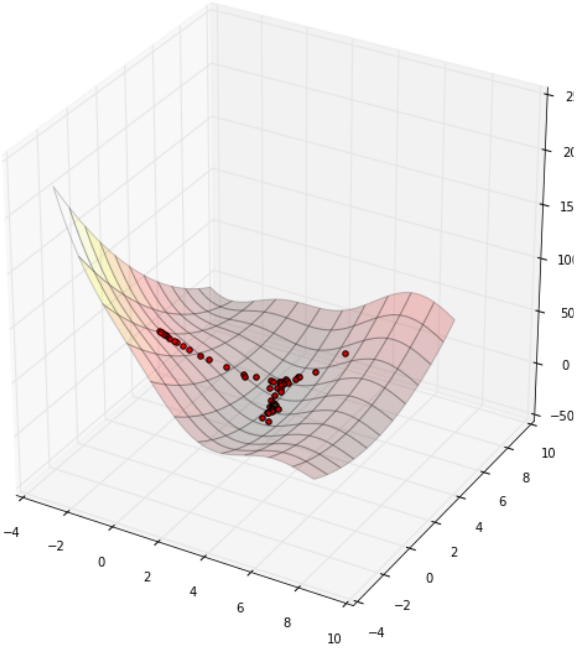
3.1 Функция *Branin*



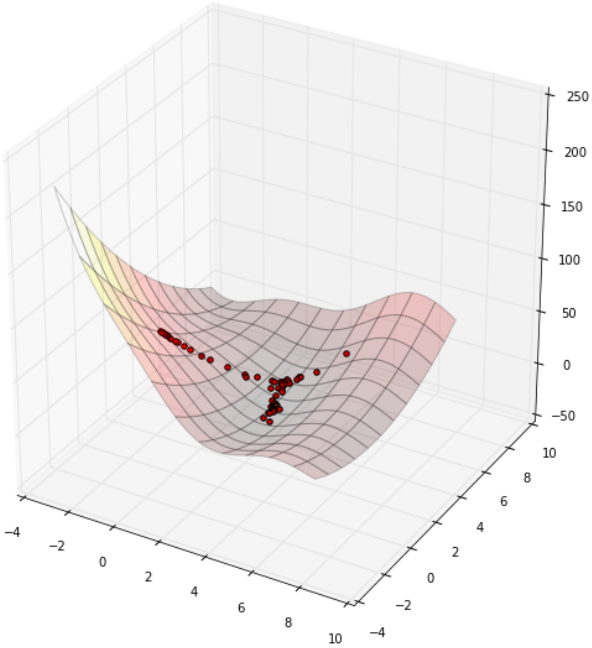
$n = 30$



$n = 50$

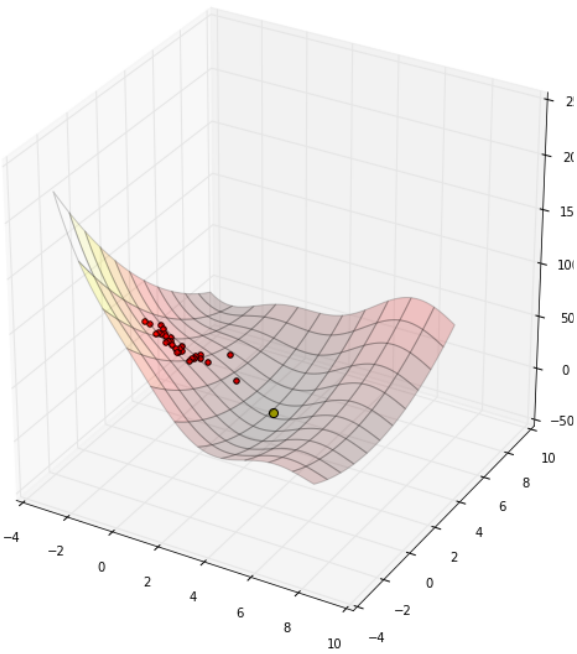


$n = 100$

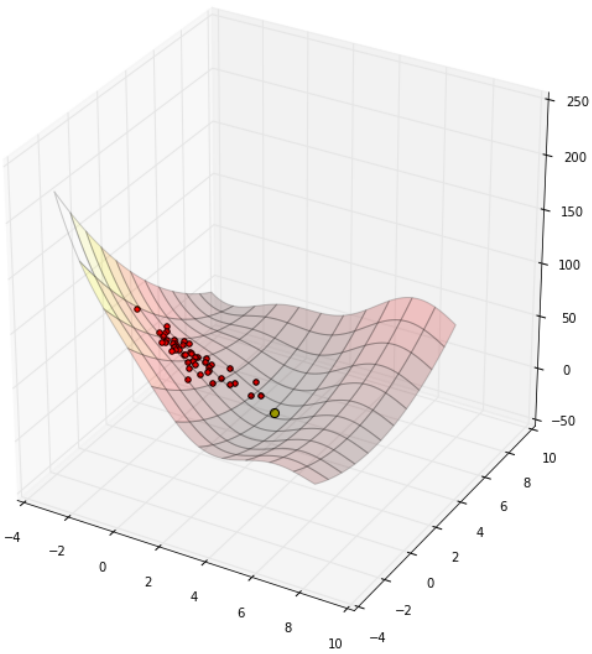


$n = 300$

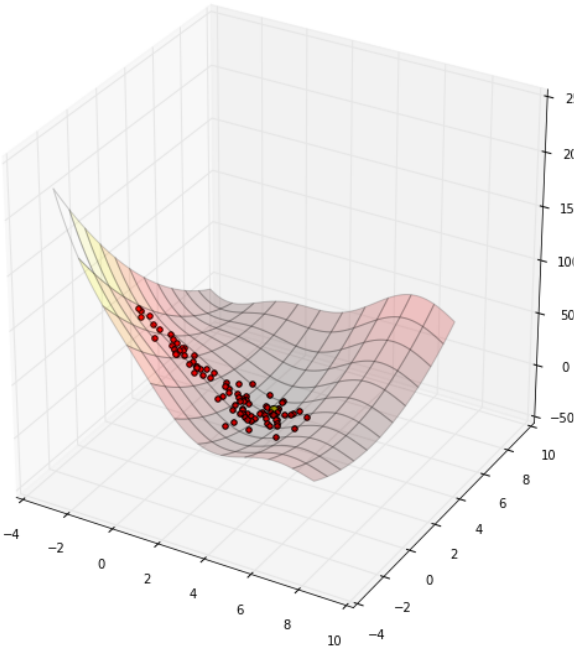
Рис. 1: Работа алгоритма *NelderMead* на функции *Branin* при разном числе шагов (n)



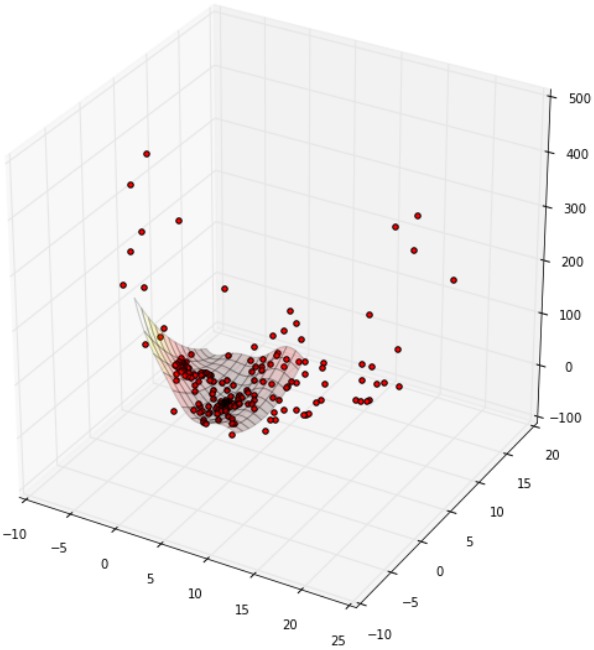
$n = 30$



$n = 50$

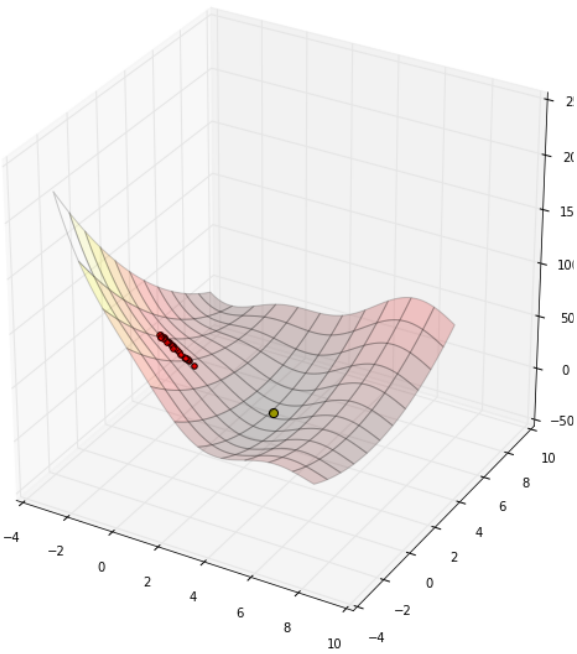


$n = 100$

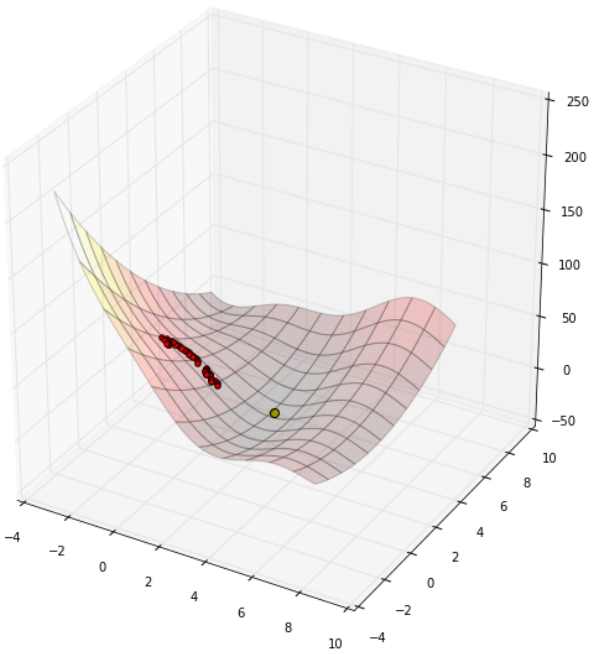


$n = 300$

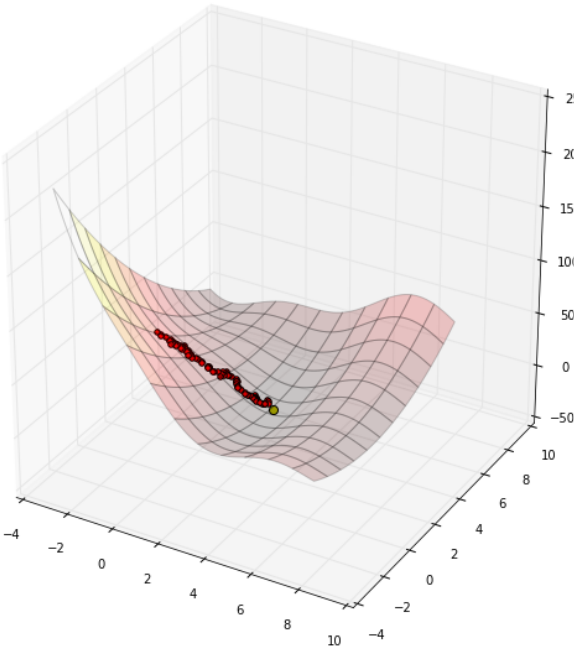
Рис. 2: Работа алгоритма $CMAES$ на функции $Branin$ при разном числе шагов (n)



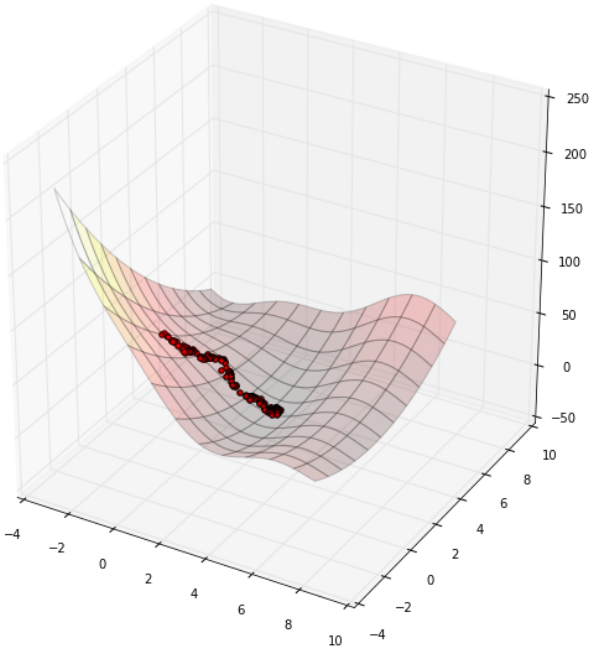
$n = 30$



$n = 50$

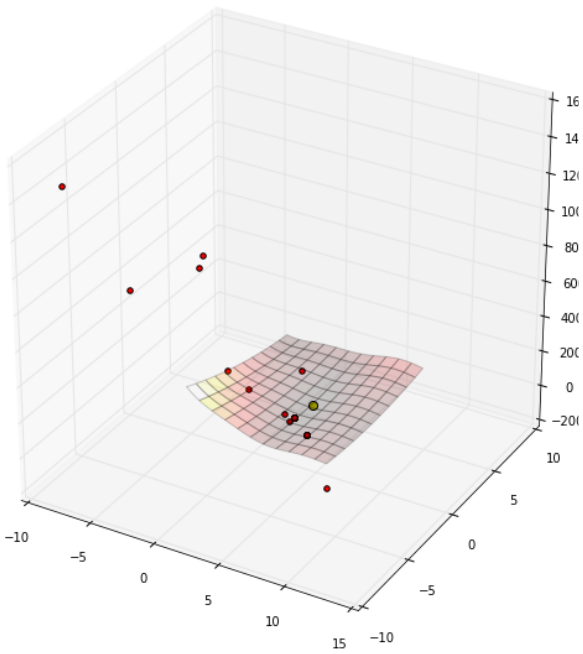


$n = 100$

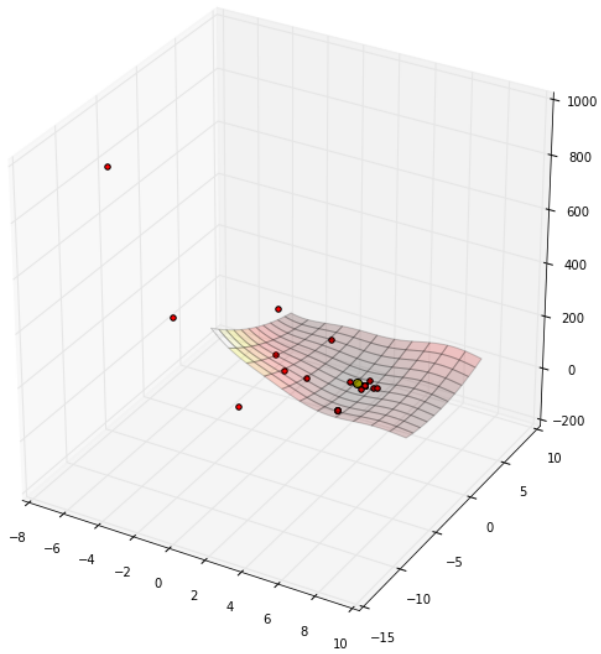


$n = 300$

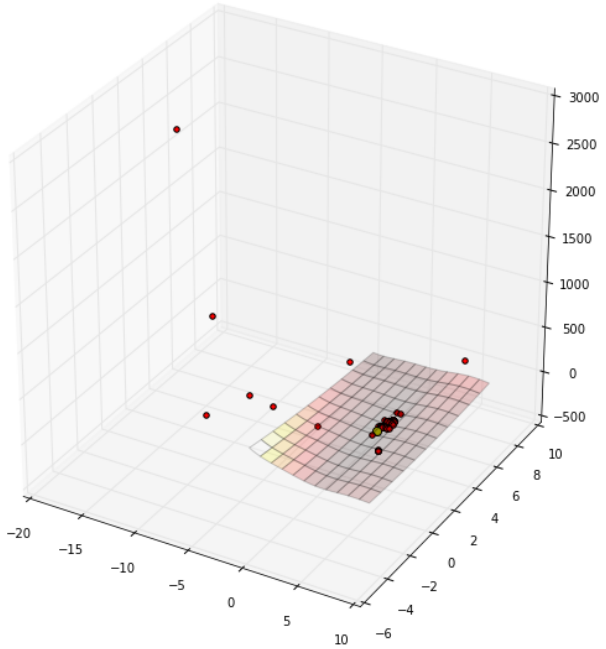
Рис. 3: Работа алгоритма *HillClimber* на функции *Branin* при разном числе шагов (n)



$n = 30$

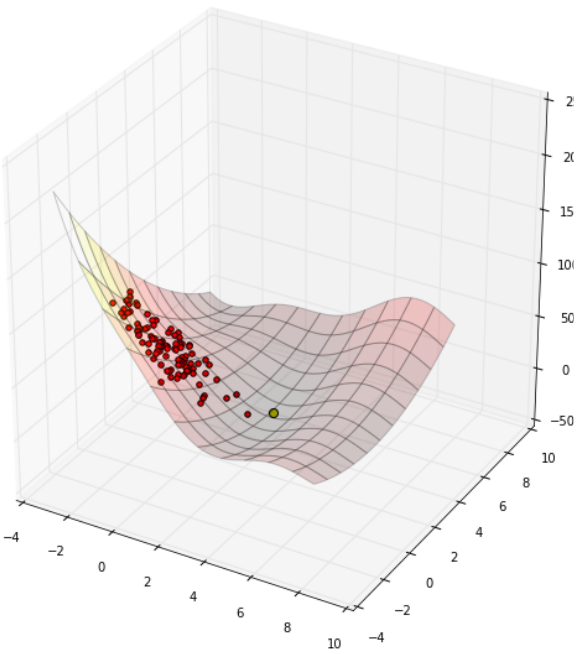


$n = 50$

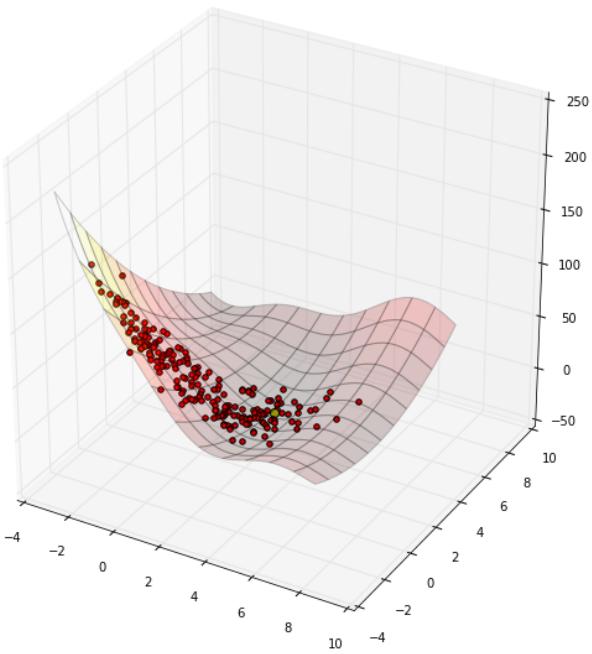


$n = 100$

Рис. 4: Работа алгоритма GA на функции $Branin$ при разном числе шагов (n)

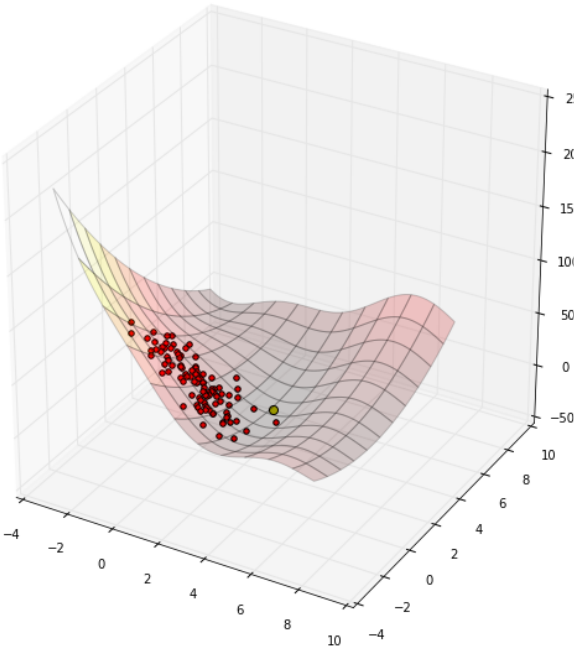


$n = 100$

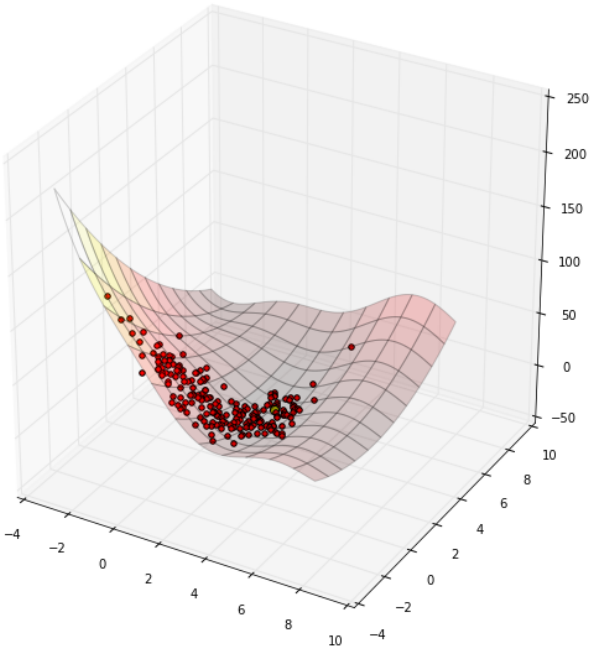


$n = 300$

Рис. 5: Работа алгоритма *OriginalNES* на функции *Branin* при разном числе шагов (n)

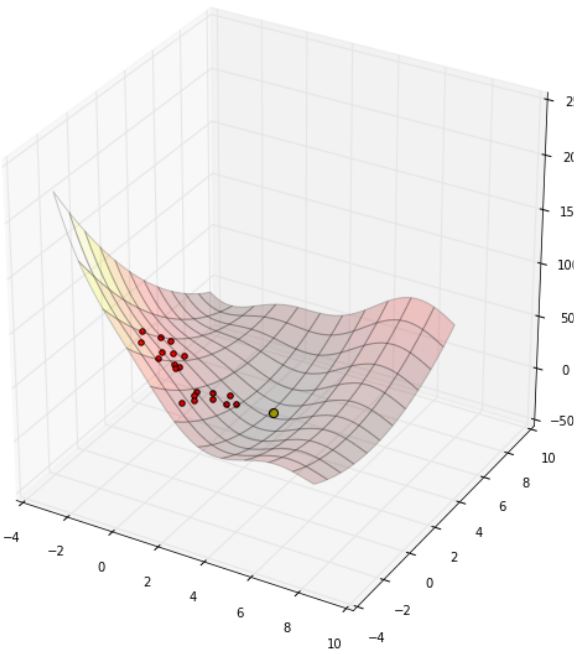


$n = 100$

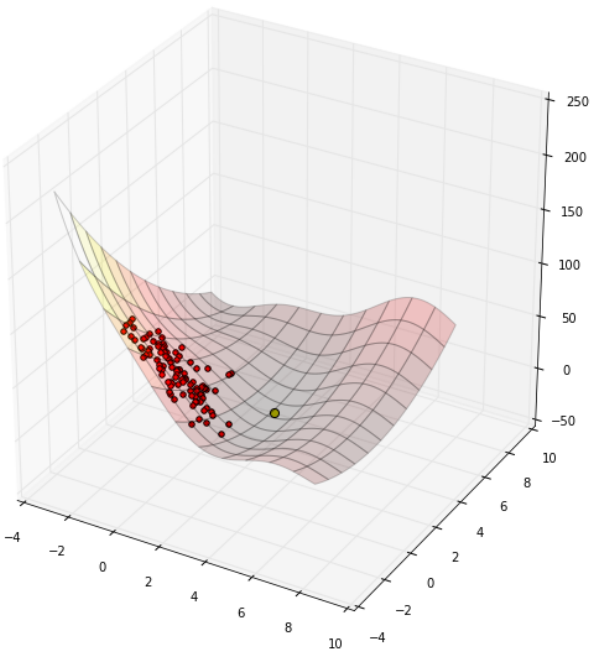


$n = 300$

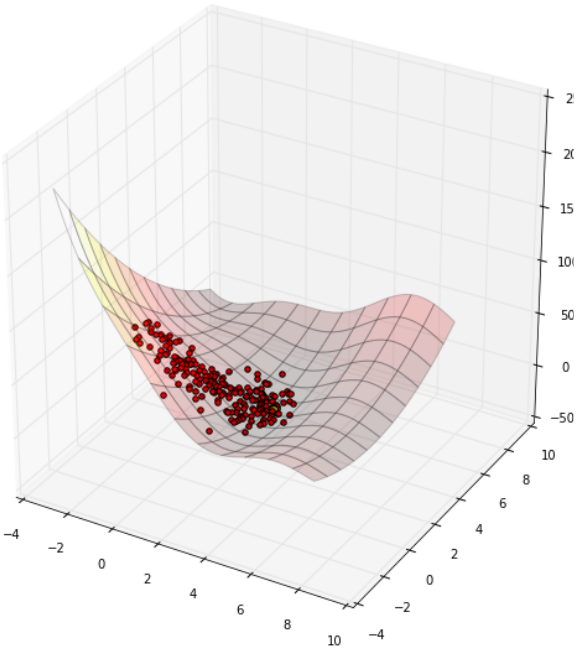
Рис. 6: Работа алгоритма *ExactNES* на функции *Branin* при разном числе шагов (n)



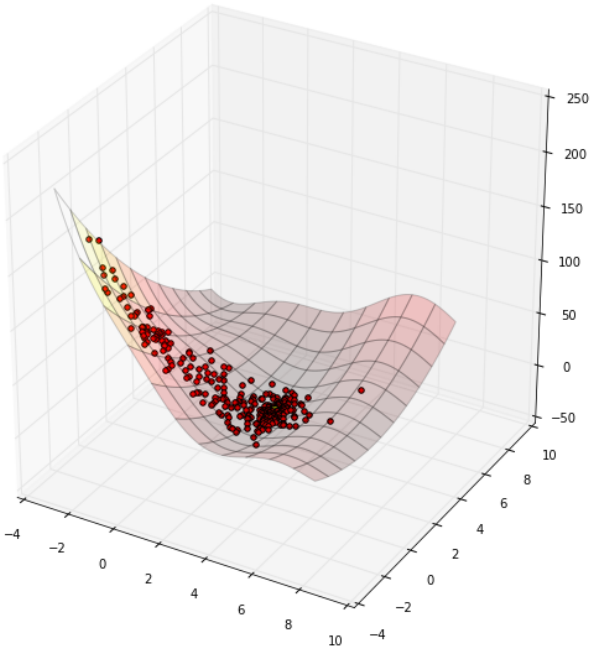
$n = 120$



$n = 200$

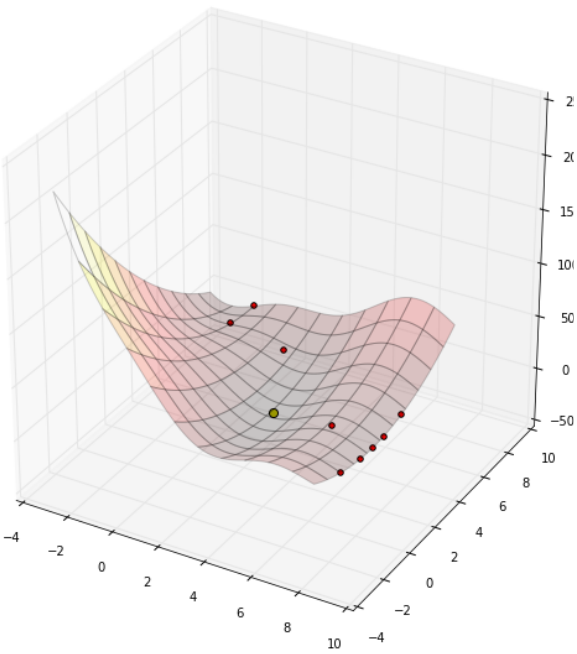


$n = 300$

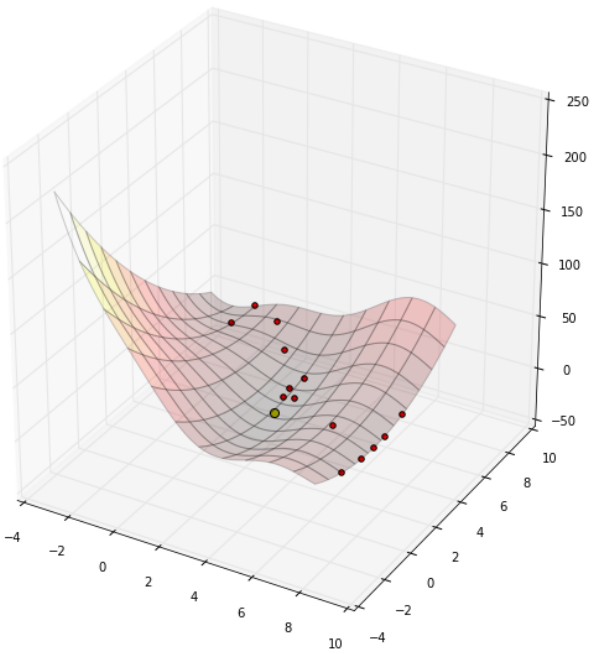


$n = 400$

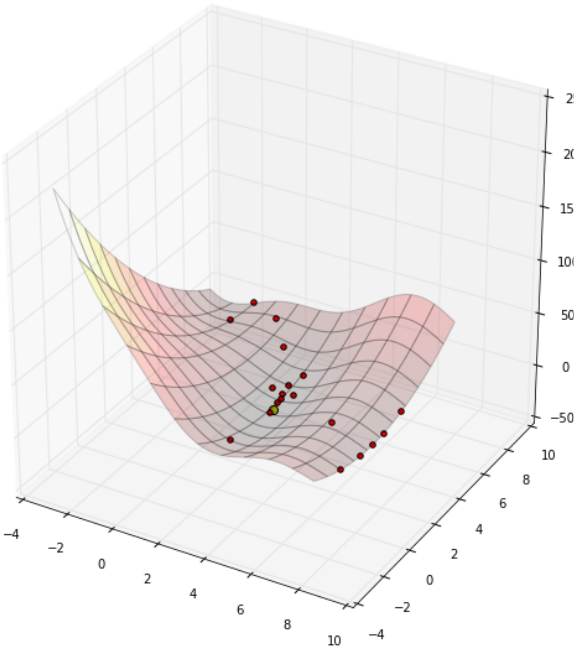
Рис. 7: Работа алгоритма FEM на функции $Branin$ при разном числе шагов (n)



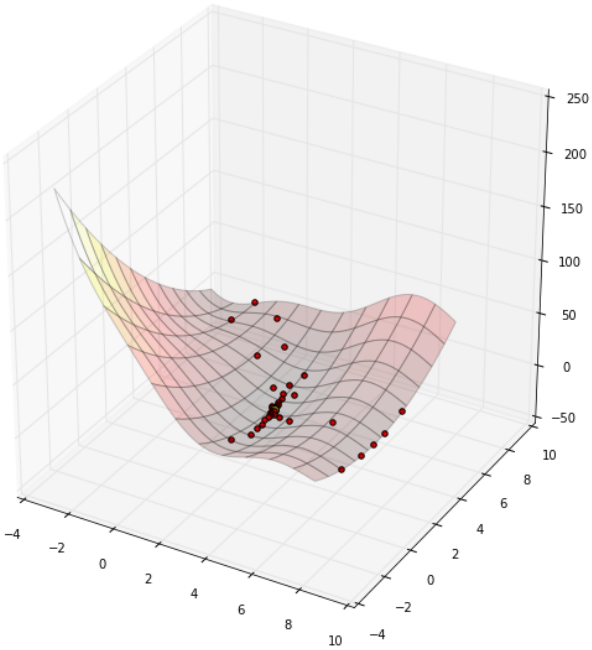
$n = 10$



$n = 15$



$n = 20$



$n = 50$

Рис. 8: Работа алгоритма *COGP* на функции *Branin* при разном числе шагов (n)

Таблица 1: Результаты сравнения на функции *Branin*

Алгоритм	Число шагов	Ошибка	Время (с)
NelderMead	30	1.6264	0.0025
NelderMead	50	0.1057	0.0042
NelderMead	100	0.0249	0.0067
NelderMead	300	0.0249	0.0065
CMAES	30	1.8810	0.0038
CMAES	50	1.1186	0.0060
CMAES	100	0.1705	0.0108
CMAES	300	0.0224	0.0362
HillClimber	30	2.9120	0.0066
HillClimber	50	2.1400	0.0057
HillClimber	100	0.4785	0.0114
HillClimber	300	0.0111	0.0363
GA	30	1.7697	0.0016
GA	50	0.6595	0.0022
GA	100	0.4013	0.0073
OriginalNES	100	1.0129	0.0080
OriginalNES	200	0.1377	0.0139
OriginalNES	300	0.3355	0.0925
OriginalNES	400	0.0955	0.1560
ExactNES	100	0.9345	0.0101
ExactNES	200	0.6540	0.0293
ExactNES	300	0.1409	0.1314
ExactNES	400	0.0738	0.2353
FEM	120	1.3896	0.0041
FEM	200	2.2477	0.0598
FEM	300	0.0843	0.1301
FEM	400	0.0541	0.1994
COGP	10	4.8381	0.2456
COGP	15	1.1506	0.4155
COGP	20	0.1317	0.6558
COGP	50	0.0485	1.9583

3.2 Функция $xsinx$

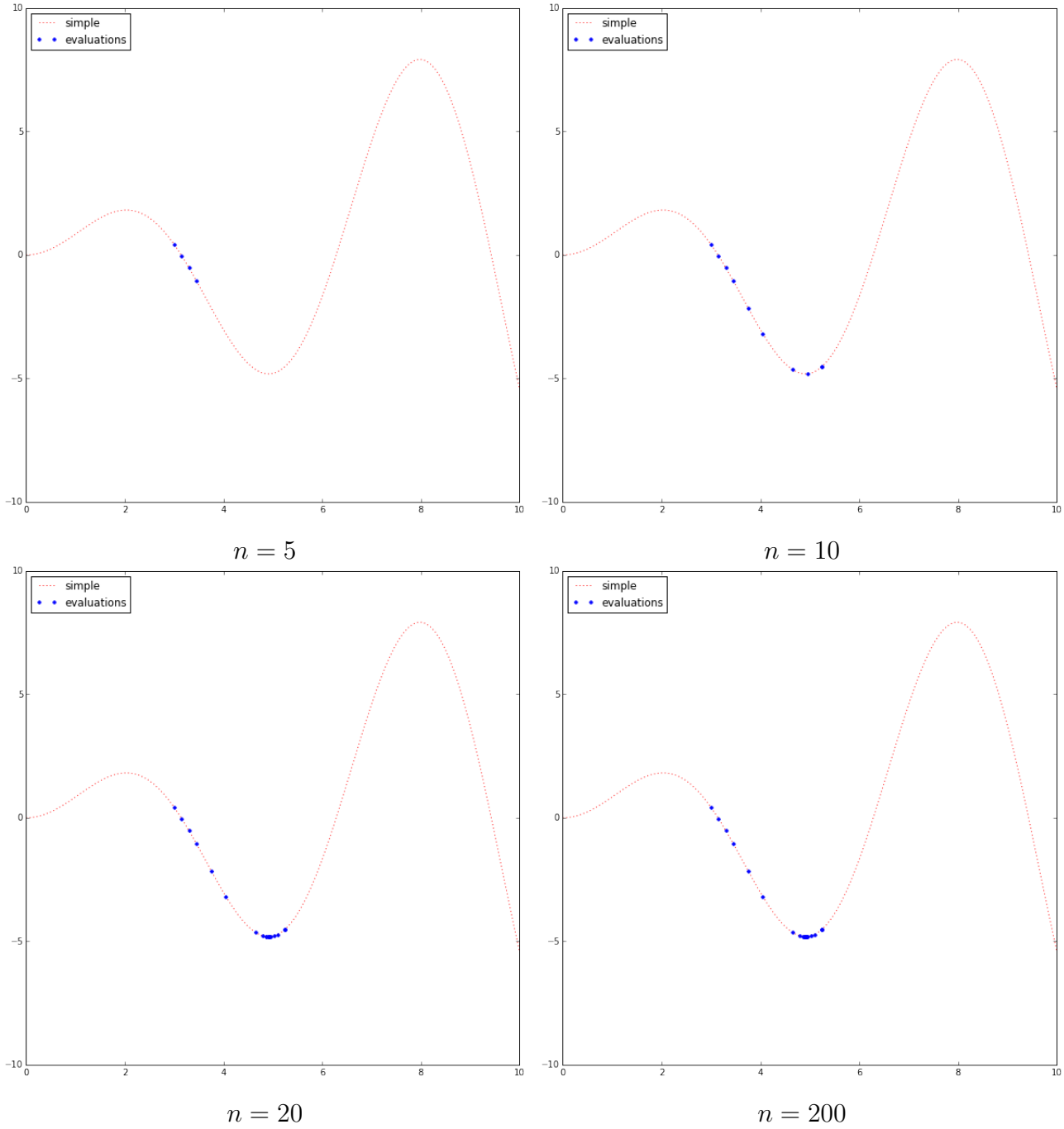
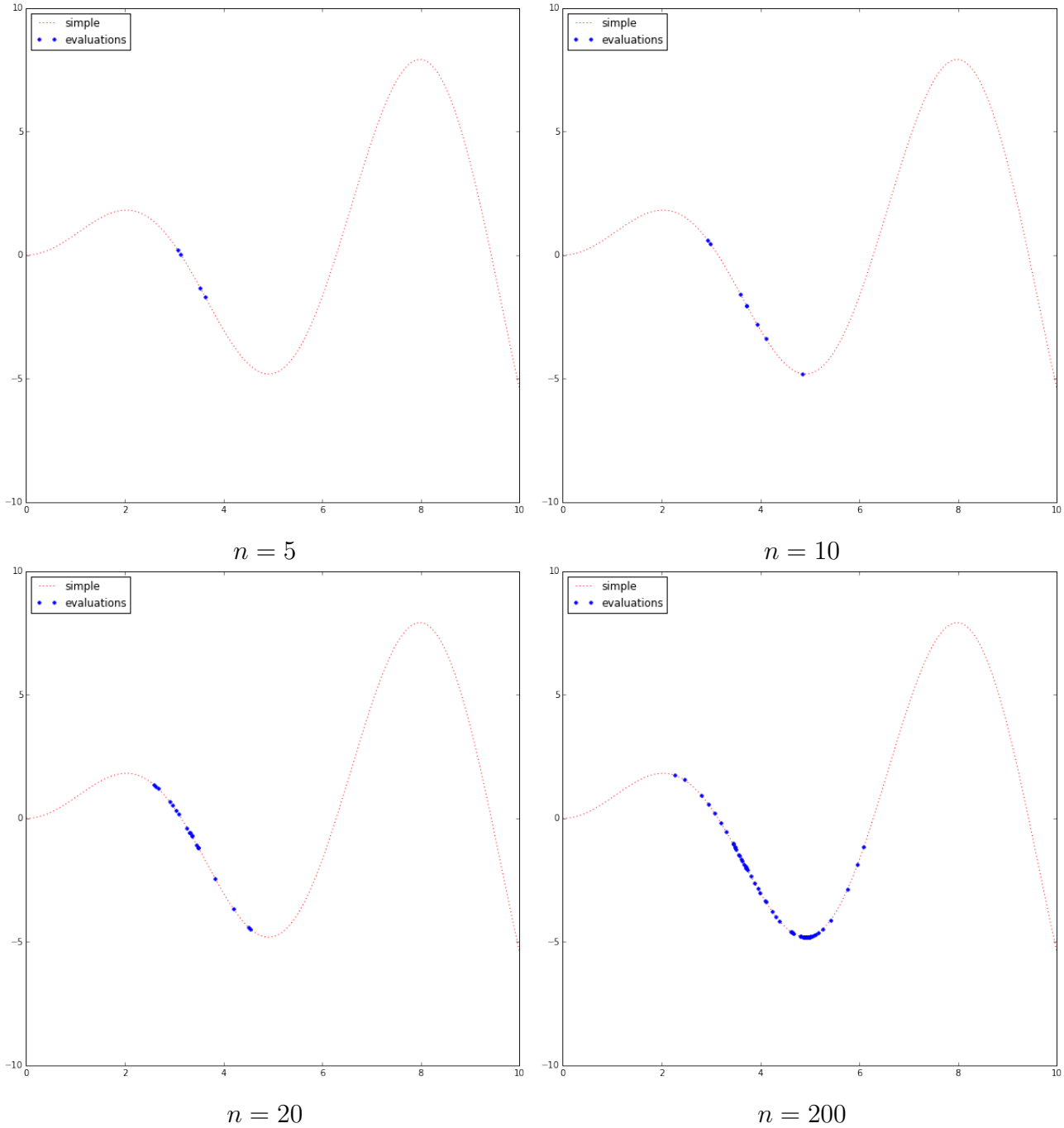


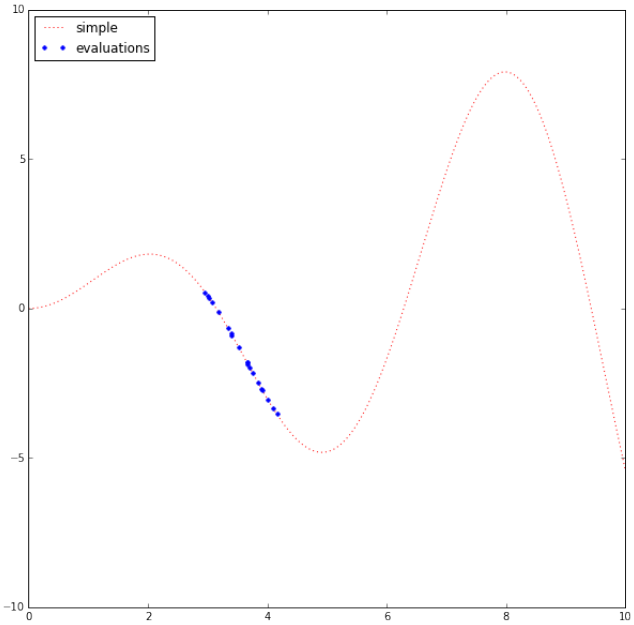
Рис. 9: Работа алгоритма $NelderMead$ на функции $xsinx$ при разном числе шагов (n)



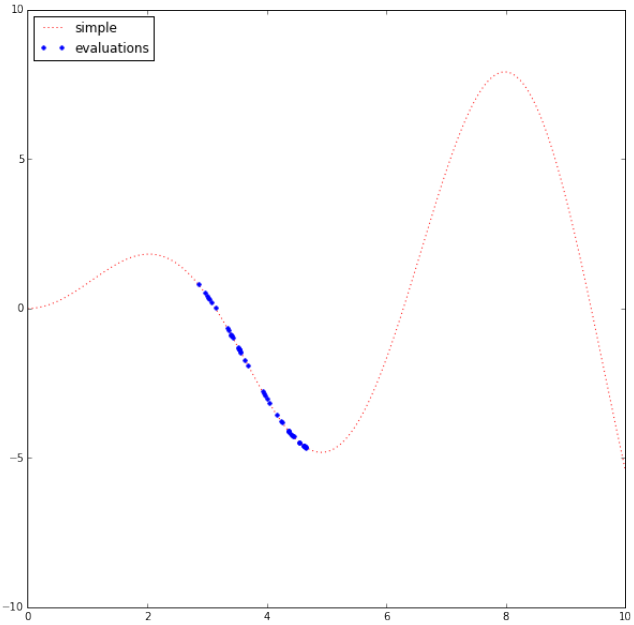
$n = 5$ $n = 10$

$n = 20$ $n = 200$

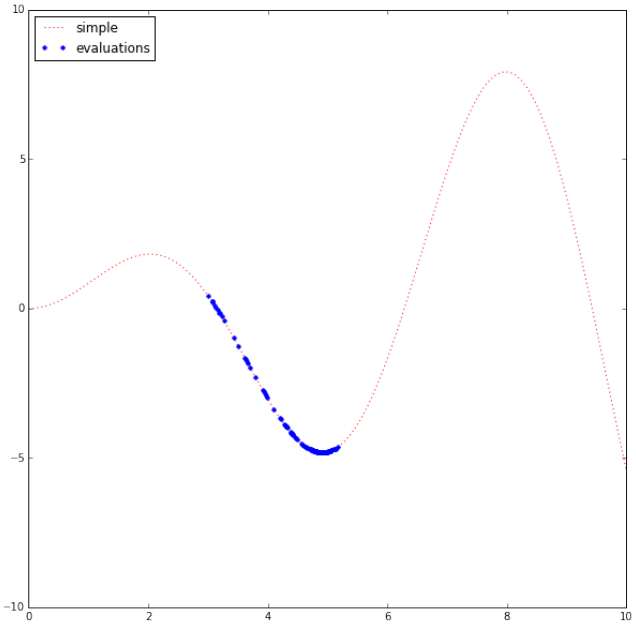
Рис. 10: Работа алгоритма $CMAES$ на функции $xsinx$ при разном числе шагов (n)



$n = 20$

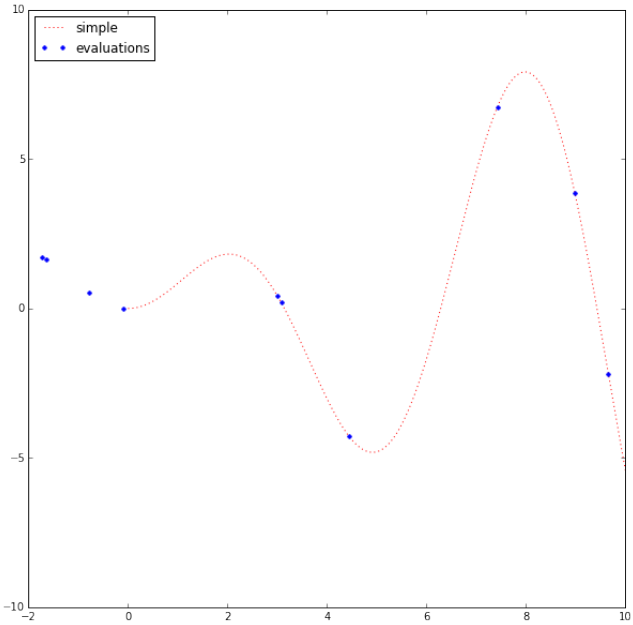


$n = 50$

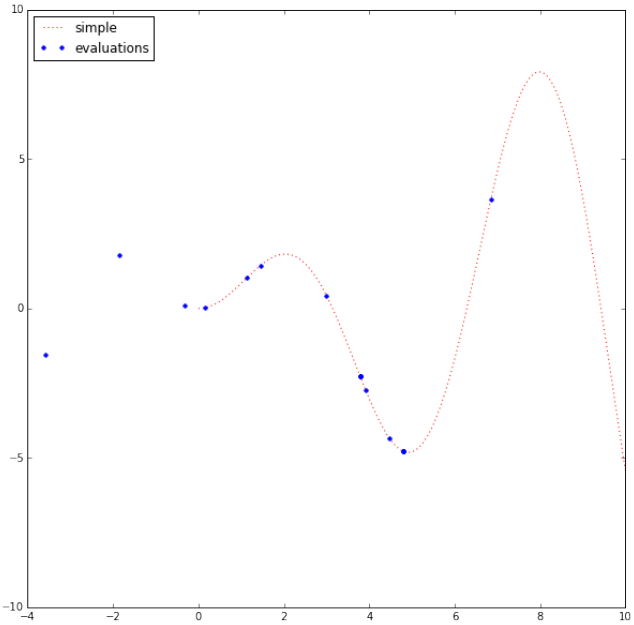


$n = 200$

Рис. 11: Работа алгоритма *HillClimber* на функции $xsinx$ при разном числе шагов (n)

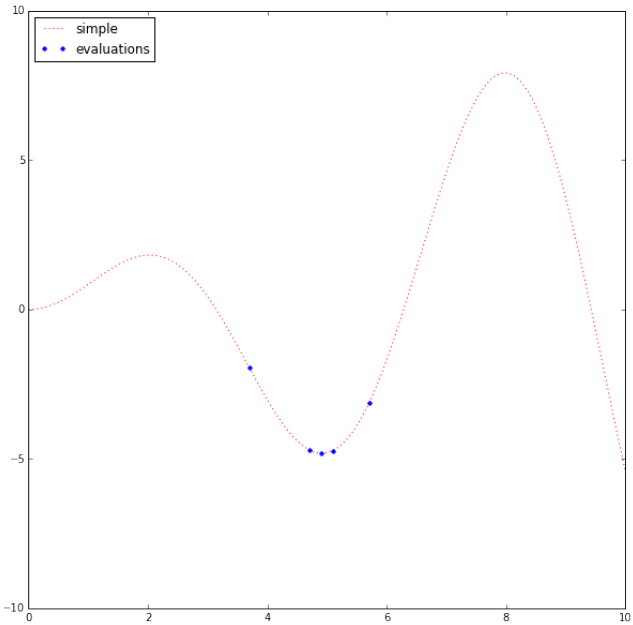


$n = 10$

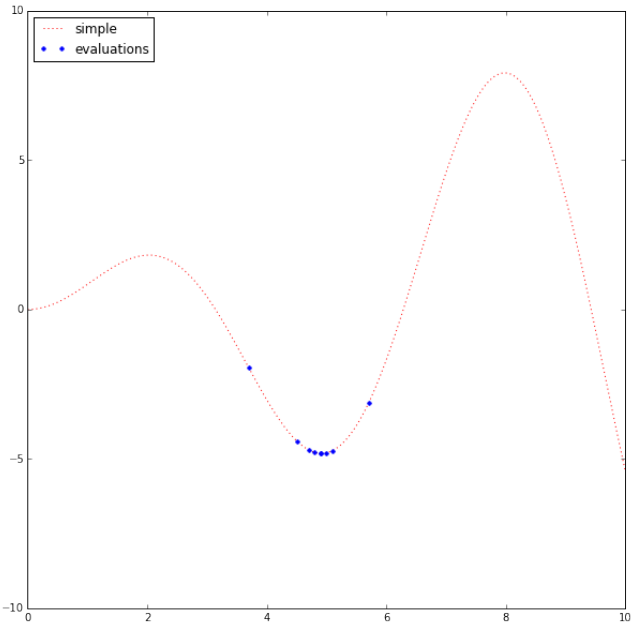


$n = 20$

Рис. 12: Работа алгоритма GA на функции $xsinx$ при разном числе шагов (n)



$n = 5$



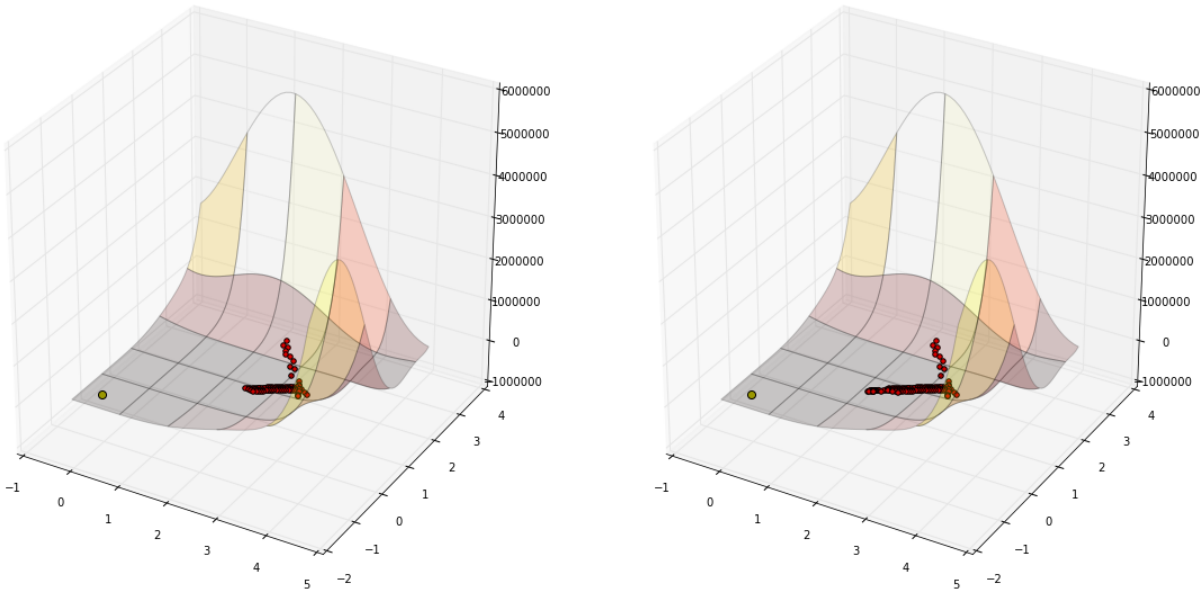
$n = 10$

Рис. 13: Работа алгоритма $COGP$ на функции $xsinx$ при разном числе шагов (n)

Таблица 2: Результаты сравнения на функции $xsinx$

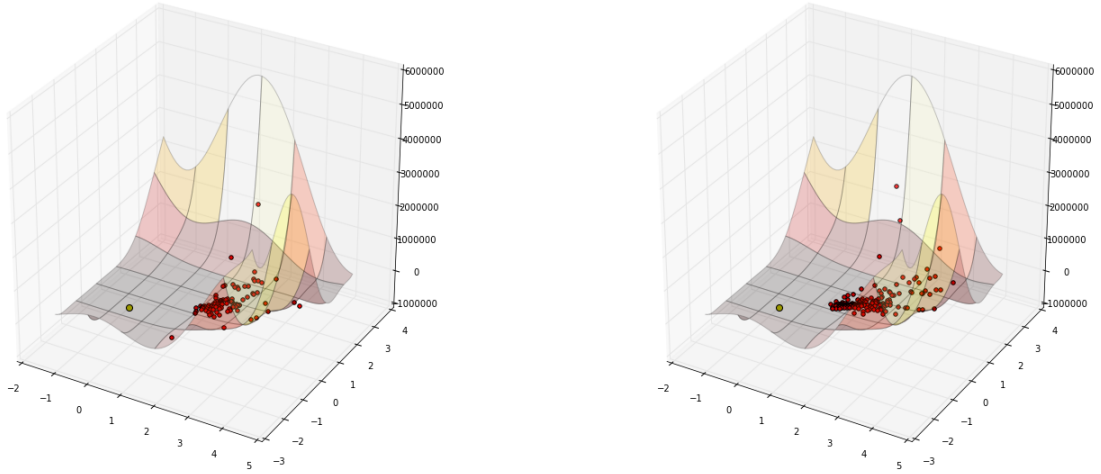
Алгоритм	Число шагов	Ошибка	Время (с)
NelderMead	5	1.4631	0.0010
NelderMead	10	0.0368	0.0014
NelderMead	20	6.8e-4	0.0018
NelderMead	200	2e-5	0.0023
CMAES	5	1.2822	0.0014
CMAES	10	0.0568	0.0017
CMAES	20	0.3716	0.0030
CMAES	200	3.5e-7	0.0307
HillClimber	20	0.7546	0.0025
HillClimber	50	0.2532	0.0050
HillClimber	200	0.0008	0.0184
GA	10	0.4740	0.0007
GA	20	0.1239	0.0008
COGP	5	0.0131	0.0422
COGP	10	0.0027	0.0732

3.3 Функция $Goldstein - Price$



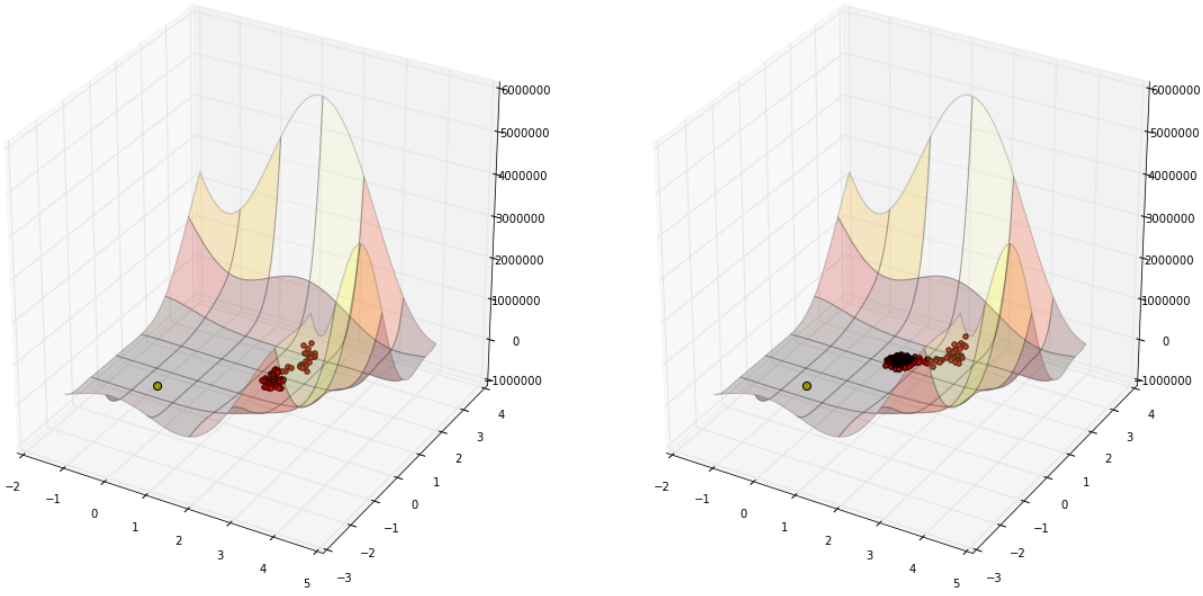
$n = 100$ $n = 600$

Рис. 14: Работа алгоритма $NelderMead$ на функции $Goldstein - Price$ при разном числе шагов (n)



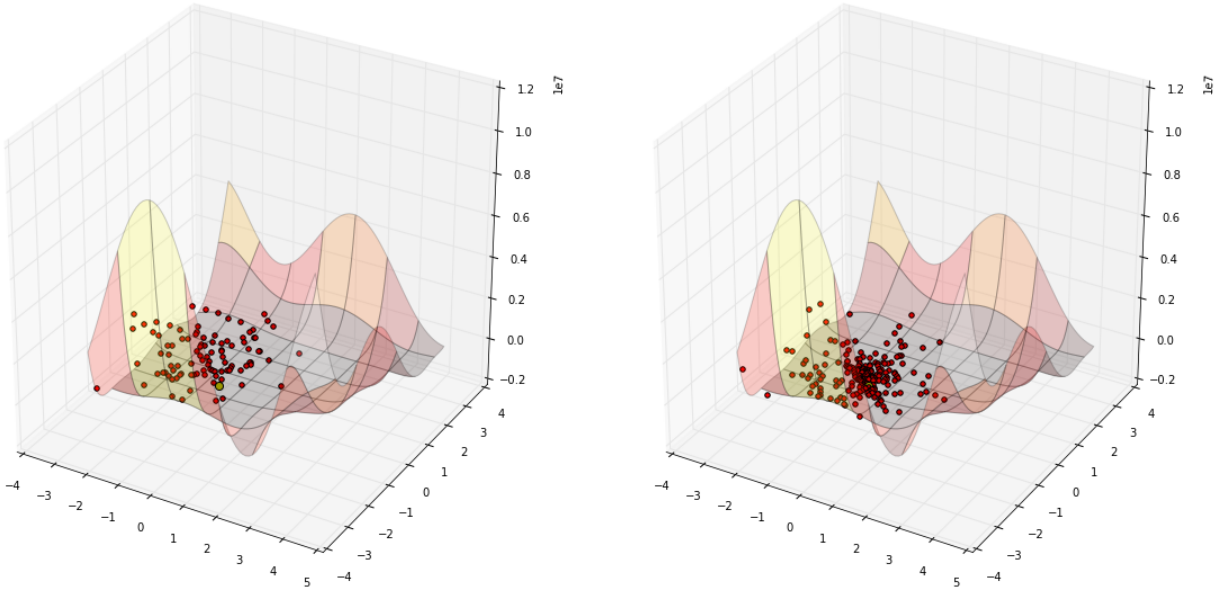
$n = 100$ $n = 600$

Рис. 15: Работа алгоритма $CMAES$ на функции $Goldstein - Price$ при разном числе шагов (n)



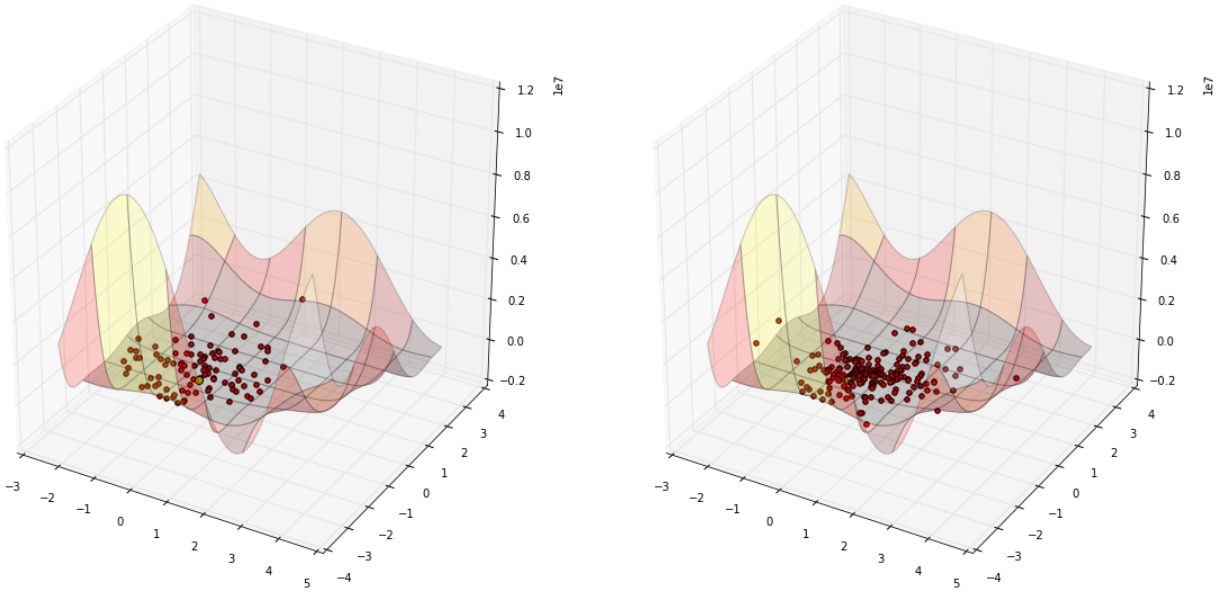
$n = 100$ $n = 600$

Рис. 16: Работа алгоритма $HillClimber$ на функции $Goldstein - Price$ при разном числе шагов (n)



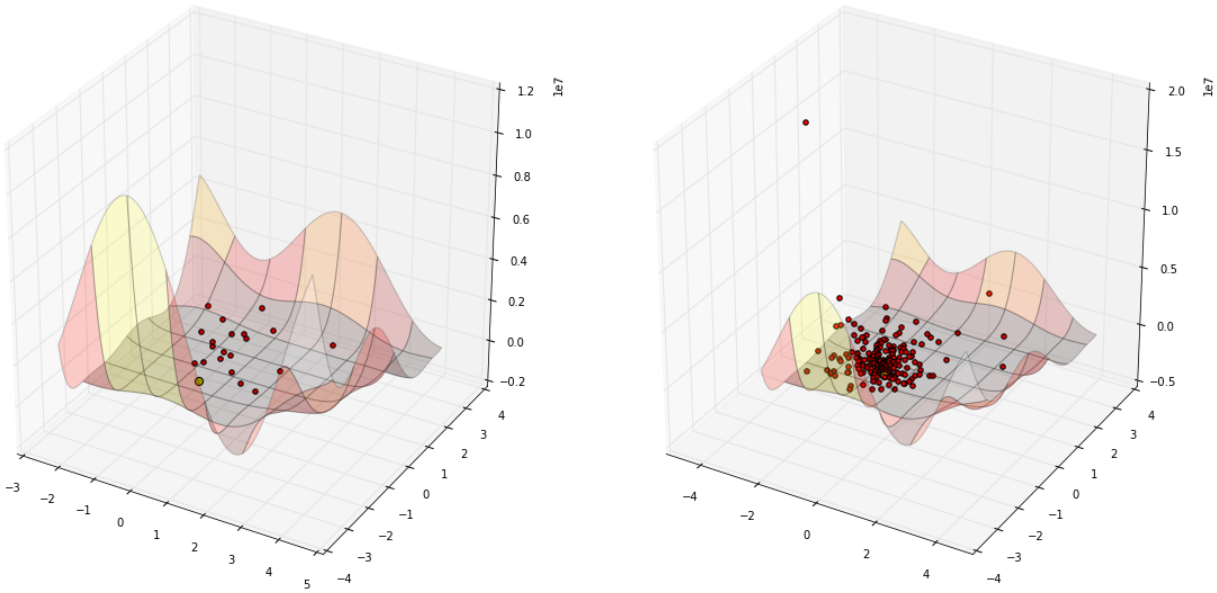
$n = 100$ $n = 300$

Рис. 17: Работа алгоритма $OriginalNES$ на функции $Goldstein - Price$ при разном числе шагов (n)



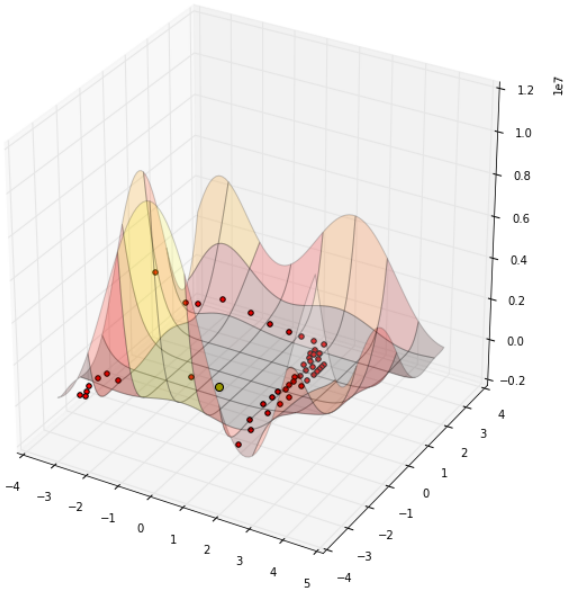
$n = 100$ $n = 300$

Рис. 18: Работа алгоритма $ExactNES$ на функции $Goldstein - Price$ при разном числе шагов (n)

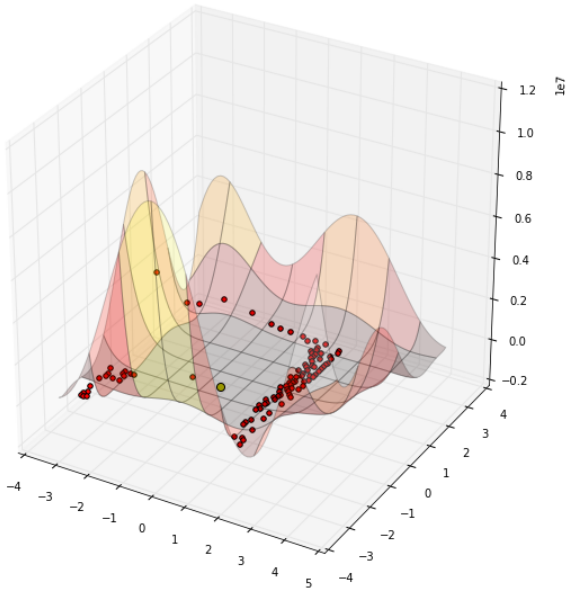


$n = 100$ $n = 300$

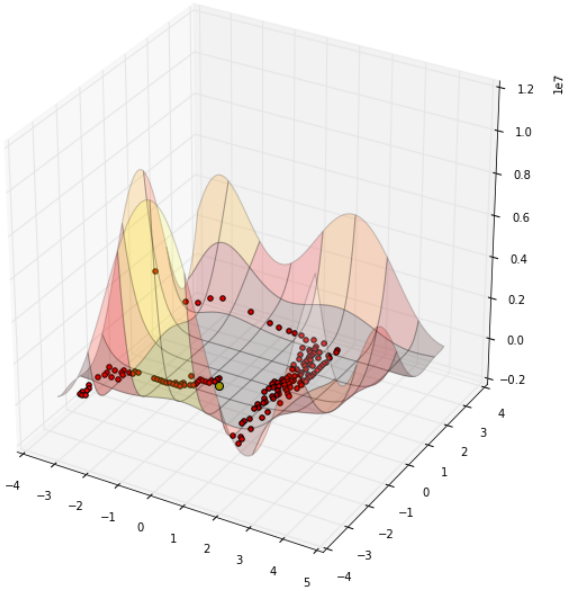
Рис. 19: Работа алгоритма FEM на функции $Goldstein - Price$ при разном числе шагов (n)



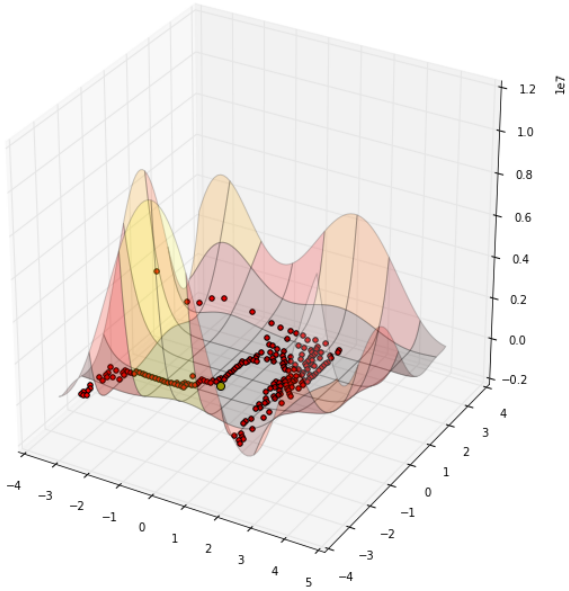
$n = 50$



$n = 100$



$n = 150$



$n = 200$

Рис. 20: Работа алгоритма *COGP* на функции *Goldstein – Price* при разном числе шагов (n)

Таблица 3: Результаты сравнения на функции $Goldstein - Price$

Алгоритм	Число шагов	Ошибка	Время (с)
NelderMead	100	2.6453	0.0094
NelderMead	600	2.1657	0.0242
CMAES	100	2.0698	0.0127
CMAES	600	2.1633	0.0940
HillClimber	100	2.3658	0.0129
HillClimber	600	2.1576	0.1157
GA	100	2.0219	0.0062
GA	600	2.4373	0.0360
OriginalNES	100	0.2239	0.0166
OriginalNES	200	0.1497	0.0173
OriginalNES	300	0.0840	0.1346
ExactNES	100	0.1139	0.0661
ExactNES	200	0.0817	0.0230
ExactNES	300	0.0897	0.2229
FEM	120	0.8597	0.0047
FEM	200	0.1149	0.0413
FEM	300	0.0388	0.1027
COGP	50	2.3021	1.1283
COGP	100	2.1633	3.9414
COGP	150	0.2236	8.9685
COGP	200	0.0236	16.5580

3.4 Функция *Colville*

Функция *Colville* - функция 4-х переменных, поэтому приводится только таблица результатов сравнения.

Таблица 4: Результаты сравнения на функции *Colville*

Алгоритм	Число шагов	Ошибка	Время (с)
NelderMead	50	2.0640	0.0434
NelderMead	90	0.8889	0.0072
NelderMead	150	0.8047	0.0242
NelderMead	600	1.6e-5	0.0470
CMAES	50	1.3407	0.0011
CMAES	90	0.8176	0.0096
CMAES	150	0.6519	0.0147
CMAES	600	0.2218	0.0745
HillClimber	50	1.5027	0.0075
HillClimber	90	0.8176	0.0100
HillClimber	150	0.6939	0.0169
HillClimber	600	0.0694	0.0652
GA	90	1.4119	0.0046
GA	150	1.2109	0.0081
GA	600	1.2032	0.0270
OriginalNES	100	1.7539	0.0105
OriginalNES	150	1.1707	0.0109
OriginalNES	600	0.4340	0.3883
ExactNES	100	2.1476	0.0142
ExactNES	150	1.8417	0.0144
ExactNES	600	0.9115	0.7729
FEM	120	1.5908	0.0062
FEM	150	1.2633	0.0124
FEM	600	0.6040	0.3303
COGP	30	0.9949	29.476
COGP	50	0.8888	73.192
COGP	90	0.0136	228.05

3.5 Вывод

Алгоритм COGP хорош в плане количества шагов до минимума. Почти на всех функциях по этому показателю он лучше остальных алгоритмов, причем в разы. К недостаткам COGP можно отнести время работы. Приближение точек гауссовским процессом и выбор

точек-кандидатов процесс довольно долгий. Это время сравнимо со временем вычисления функции в точке. Из-за этих вычислительных расходов алгоритм серьезно уступает по времени всем другим алгоритмам. На функциях одной и двух переменных время COGP хуже в 10-100 раз, чем у остальных, но в целом это десятые и сотые доли секунды. На функциях трех, четырех переменных COGP хуже примерно в 100-1000 раз по времени, и это уже секунды и десятки секунд. Можно отметить, что для функции *Goldstein-Price* алгоритм CMAES, который хорошо себя проявил на других функциях, очень плохо сходится к минимуму, тогда как COGP сходится хорошо.

4 Использование COGP для оптимизации grid search

4.1 Схема работы grid search

Grid search («поиск на сетке») - алгоритм машинного обучения, решающий задачу выбора оптимальных параметров (гиперпараметров) из множества (сетки) для обучаемого алгоритма (learning algorithm). *Grid search* - это просто поиск перебором по вручную заданному подмножеству множества гиперпараметров обучаемого алгоритма. *Grid search* снабжается *метрикой*, которая показывает насколько обучаемый алгоритм с такими параметрами хорош. Для каждого элемента подмножества *grid search* оценивает значение метрики. В качестве результата работы алгоритм выдает оптимальные гиперпараметры (гиперпараметры с наибольшим значением метрики).
В качестве примера можно привести SVM классификатор с RBF ядром. Он имеет, по крайней мере, 2 гиперпараметра: C - константа регуляризации, γ - параметр ядра. Примером сетки на входе алгоритма *grid search* может быть:

$$C \in \{10, 100, 1000\}; \gamma \in \{0.1, 0.2, 0.5, 1.0\} \tag{3}$$

В процессе работы *grid search* тренирует SVM классификатор на данных с каждой парой (C, γ) декартового произведения двух множеств, оценивает качество работы с использованием метрики.
Согласно [8], к недостаткам можно отнести значительное ухудшение работы при увеличении размерности пространства гиперпараметров. Однако, этот алгоритм легко параллелится, так как обучения классификатора при разных параметрах не зависят друг от друга.

4.2 Оптимизация с использованием COGP

Алгоритм *grid search* перебирает все точки сетки, для каждой из точек тренирует обучаемый алгоритм на обучающей выборке, затем на тестовой выборке оценивает метрику. Это не очень эффективно, так как при большом наборе данных процесс тренировки и оценки метрики занимает значительное время. Встает вопрос оптимизации алгоритма «поиск на сетке». В работе рассматривается оптимизация, которая позволяет не делать эту затратную операцию в каждой точке сетки, а только в минимальном количестве точек.
Рассмотрим процесс работы алгоритма *grid search* немного по-другому. Тренировку обучаемого алгоритма на обучающей выборке данных при фиксированных гиперпараметрах и оценку метрики обученного алгоритма на тестовых данных можно понимать как вычисление *функции*, которая принимает в качестве аргументов гиперпараметры и которая возвращает оценку метрики. Нас интересуют гиперпараметры, при которых метрика будет максимальной. Таким образом, оптимизация алгоритма *grid search* представляет собой поиск максимума этой *функции*. Поиск максимума можно осуществить, используя какой-либо алгоритм оптимизации «черного ящика».

4.3 Сравнение работы с классическим grid search

В работе в качестве алгоритма для поиска максимума был выбран реализованный COGP. COGP был встроен в классический (обычный) алгоритм *grid search*, реализация которого была взята из библиотеки для машинного обучения Scikit-Learn.
При сравнении использовалось два набора данных:

- digits (1797 элементов, 10 классов, 64 признака);
- данные Хиггса (250000 элементов, 2 класса, 30 признаков).

Использовалось два классификатора:

- SVC
- AdaBoost с базовым классификатором DecisionTree

Результаты были получены на компьютере Intel Core i5 2.67GHz 4Gb RAM. Они оформлены в виде таблиц (табл. 5-9).

Таблица 5: данные: $digits$, классификатор: SVC, сетка: 8×3

Алгоритм	Число шагов	Время (с)
Обычный	24	20.62
Оптимизированный	10	16.68

Таблица 6: данные: $digits$, классификатор: $AdaBoost$, сетка: 9×7

Алгоритм	Число шагов	Время (с)
Обычный	63	25.10
Оптимизированный	9	3.35

Таблица 7: данные: $Higgs$, классификатор: $AdaBoost$, сетка: 9×5

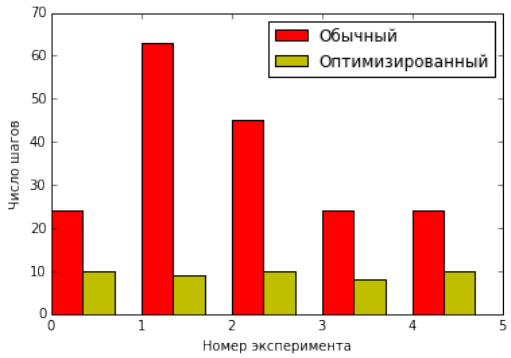
Алгоритм	Число шагов	Время (с)
Обычный	45	4397.7
Оптимизированный	10	1543.8

Таблица 8: данные: выборка (1000 элементов) из $Higgs$, классификатор: SVC, сетка: 8×3

Алгоритм	Число шагов	Время (с)
Обычный	24	9.12
Оптимизированный	8	4.53

Таблица 9: данные: выборка (10000 элементов) из $Higgs$, классификатор: SVC, сетка: 8×3

Алгоритм	Число шагов	Время (с)
Обычный	24	963.7
Оптимизированный	10	452.2



4.4 Вывод

Проделанная оптимизация очень хорошо уменьшает время работы алгоритма. Время работы оптимизированного алгоритма в 2-8 раз меньше, чем у обычного. При этом считаются значения в 2-6 раз меньшем количестве точек. Такого хорошего результата удается достичь благодаря тому, что «вычисление функции» (обучение классификатора, оценка метрики) делается довольно долго (намного дольше, чем вычисления на каждом шаге в алгоритме COGP). Становится важным число таких вычислений (чем меньше — тем лучше). Число шагов до достижения экстремума — одно из главных преимуществ COGP, как было видно в предыдущей главе. Поэтому сделанная оптимизация так эффективна.

Однако, обычный *grid search* может противопоставить то, что он просто параллелится, и с использованием вычислений на разных компьютерах будет работать быстрее. Алгоритм *grid search* с оптимизацией наоборот, распараллелить очень проблематично, так как на каждом шаге он выбирает по одной точке, пользуясь точками, посчитанными на предыдущих шагах. Возможно, продолжение данной работы будет посвящено как раз этой проблеме.

5 Результат работы оптимизации на данных Хиггса

5.1 Higgs Boson Machine Learning Challenge

Higgs Boson Machine Learning Challenge — самое крупное международное соревнование по анализу данных и машинному обучению. Соревнование уже завершилось. Оно проводилось совместно с ЦЕРНом и посвящено поиску Бозона Хиггса, используя данные с эксперимента ATLAS. Для участия не нужно знать физику элементарных частиц. Цель соревнования — использовать данные эксперимента для того, чтобы классифицировать все события в эксперименте на 2 группы: сигнал (распад Бозона Хиггса, S) и не сигнал (фон, шум, B). Для оценки результатов классификации вводится специальная метрика — AMS (approximate median significance):

$$AMS = \sqrt{2((s + b + b_r)log(1 + \frac{s}{b + b_r}) - s)}, \tag{4}$$

где

$$b_r = 10, s = \sum w_i 1\{y_i = S\}1\{\hat{y}_i = S\}, b = \sum w_i 1\{y_i = B\}1\{\hat{y}_i = S\},$$

$(y_1, \cdots, y_n) \in \{B, S\}^n$ - вектор истинных значений, $(\hat{y}_1, \cdots, \hat{y}_n) \in \{B, S\}^n$ - вектор предсказанных значений, $(w_1, \cdots, w_n) \in \mathbb{R}^{+n}$ - вектор заданных весов.
Задача соревнования состоит в том, чтобы не просто классифицировать все события на две группы, а чтобы также максимизировать AMS метрику.

5.2 Оптимизация метрики AMS

В работе была произведена попытка запустить оптимизированный алгоритм *grid search* на данных Хиггса и посмотреть, какой метрики AMS можно достичь. В качестве классификатора был выбран AdaBoost. Рассматривалась сетка, состоящая из элементов ($n_estimators$, $learning_rate$, $threshold$), где $n_estimators$ и $learning_rate$ — гиперпараметры классификатора, а $threshold$ — порог для различения сигналов на S и B (влияет на метрику AMS). Тем самым, в результате работы алгоритма должны быть получены оптимальные гиперпараметры для классификатора и оптимальный порог, с которым метрика AMS максимальна.

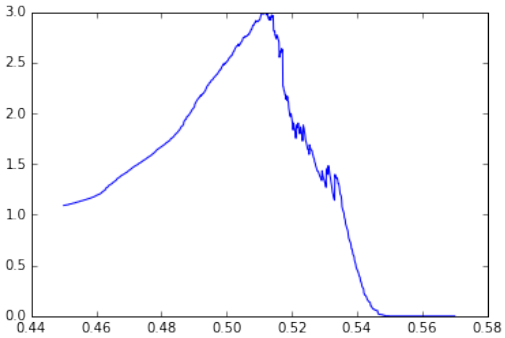


Рис. 21: Зависимость AMS от $threshold$ при оптимальных гиперпараметрах

Результат работы алгоритма: $n_estimators = 70$, $learning_rate = 0.3$, $threshold = 0.512$. Значение AMS при этом: $AMS = 2.99049$.
Для сравнения: результат человека на первом месте $AMS = 3.80581$. Таким образом, полученный в работе результат нельзя назвать очень успешным. Но, возможно, для достижения большего успеха нужно применять и комбинировать другие методы.

6 Заключение

В данной работе был изучен и реализован алгоритм *blackbox* оптимизации COGP, основанный на гауссовских процессах. Было проведено сравнение этого алгоритма с другими известными алгоритмами. Можно сказать, что COGP уверенно смотрится рядом с остальными алгоритмами. Есть недостаток — это время работы. На функциях одной и двух переменных время COGP хуже в 10-100 раз, по сравнению с остальными, но в целом это — десятые и сотые доли секунды. На функциях трех, четырех переменных COGP хуже примерно в 100-1000 раз по времени, и это уже секунды и десятки секунд. Однако, у COGP есть и серьезное преимущество — число шагов до нахождения экстремума. Алгоритм делает, в среднем, в 2-15 раз меньше шагов до его достижения.

Также, в работе была проведена оптимизация алгоритма машинного обучения *grid search* (при помощи алгоритма COGP). Именно здесь оказалось чрезвычайно полезным главное преимущество COGP. По результатам сравнения можно сказать, что оптимизация достаточно эффективна. Время работы оптимизированного алгоритма в 2-8 раз меньше, чем у обычного. При этом считаются значения в 2-6 раз меньшем количестве точек. Существует проблема распараллелить оптимизированный алгоритм. Возможно, этому будет посвящена дальнейшая работа.

Список литературы

[1] T. Schaul, *Studies in Continuous Black-box Optimization*, Technische Universität München (2011).

[2] К. В. Воронцов. *Курс лекций по машинному обучению*, http://www.machinelearning.ru/ (2014)

[3] C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning*, the MIT Press (2006).

[4] Y. Zhang, W. Xu, and J. Callan. *Exploration and exploitation in adaptive filtering based on bayesian active learning*, 896-903, ICML (2003).

[5] M. Molga, C. Smutnicki. *Test functions for optimization needs* (2005).

[6] D. R. Jones. *A taxonomy of global optimization methods based on response surfaces*, Journal of Global Optimization (2001).

[7] C. Adam-Bourdariosa, G. Cowan. *Learning to discover : the Higgs boson machine learning challenge* https://www.kaggle.com/c/higgs-boson (2014)

[8] J. Bergstra, Y. Bengio. *Random Search for Hyper-Parameter Optimization.* J. Machine Learning Research 13 (2012).